Streaming Computational Science: Applications, Technology and Resource Management for HPC.

Geoffrey C. Fox*, Devarshi Ghoshal‡, Shantenu Jha†, Andre Luckow†, Lavanya Ramakrishnan‡,
* School of Informatics and Computing, Indiana University, Bloomington, IN 47408, USA
Email: gcf@indiana.edu
† Electrical and Computer Engineering, RADICAL, Rutgers University, Piscataway, NJ 08854, USA
Email: shantenu.jha@rutgers.edu
‡ Lawrence Berkeley National Laboratory
Email: lramakrishnan@lbl.gov

Data streaming from on-line instruments, large scale simulations, and distributed sensors such as those found in transportation systems and urban environments point to the growing interest and important role of streaming data and related real-time steering and control.

We define a stream to be a possibly unbounded sequence of events. Successive events may or may not be correlated and each event may optionally include a timestamp. Exemplars of streams include time-series data generated by instruments, experiments, simulations, or commercial applications including social media posts. Steering is defined as the ability to dynamically change the progression of a computational process such as a large-scale simulation via an external computational process.

Steering, which is inevitably real-time, might include changing progress of simulations, or realigning experimental sensors, or control of autonomous vehicles. Streaming and steering often occur together. An example could be for an exascale simulation where it is impractical to store every timestep and the data must be reduced, resulting in streams which may constitute the final results from the simulation in a manner similar to the way we use data from an instrument in a massive physics experiment.

The Department of Energy (DOE) Office of Science and National Science Foundation (NSF) facilities including accelerators, light sources, neutron sources and environmental sensors are producing large volumes of streaming data. The streaming data needs to be analyzed in reactive real-time, or processed in near real-time to enable next- generation scientific discoveries. There has also been an explosion of new research and technologies for stream analytics from the academic and private sectors to address the growing data volumes from social media and other web applications. However, there has been no effort in either documenting the critical research opportunities or building a community that can create and foster productive collaborations across scientific disciplines, government agencies and industry.

To address these shortcomings, a two-part workshop series, STREAM: Streaming Requirements, Experience, Applications and Middleware Workshop (STREAM2015 and STREAM2016), was conducted to bring the community together as well as to identify gaps and future efforts needed across various funding agencies. This paper summarizes the lessons learnt from these twin workshops and some selective research examples consistent with workshop recommendations.

I. Application Characteristics

Several characteristics were identified that should be used to develop a detailed classification of streaming applications and in deriving requirement for processing system. These include the size of events; the use of control or steering; the connection between events such as their ordering and stateful event processing; use of human in the loop for feedback; universality of interfaces; adaptive pipelines needed in research today; need to identify the important information "rapidly"; access control; adaptive flow control; the challenge of real-time data assimilation; complexity of data in individual events; need for fault tolerance; provenance especially in adaptive applications where data result of previous workflow; accuracy and use of sampled data; error recovery problematic data will choke when passed through the algorithm again; what are data structures and appropriate storage matching hardware and application needs. A Table can be found in the Reports [1] which is not reproduced here for space considerations.

In most cases, the data is distributed, which affects synchronization, parallelism and algorithms. Synchronization might be needed to produce for example a coherent state, but this is hard with inherently distributed asynchronous data sources and processing. It is useful to understand the latency that can be tolerated between data collection and processing data.

Coupled streams, multiple streams and interaction of streams with distributed data are important in some cases. The response time is important for UAV applications and robots; it is not so critical for e-commerce transaction streams. The needed data longevity is unclear – some fields assume data is kept all the time but where do you store it and what is the cost can become dominant considerations. We need to characterize the value of data how much data are we willing to lose during the processing?

Application processing can be characterized by the complexity of processing, the possible need for a quick turnaround, offline or online mode etc. We need to distinguish between two types of streaming applications: firstly, a closed process that runs for a long-time, and secondly an adaptive computation with lot of human involvement. The latter is seen in processing of experimental data with adjustments to data-gathering equipment based on analysis of results. Some science applications require high-speed, fully automatic and complex adaptive processing. Here it is often not cost-effective to make an automated pipeline for each case.

Within large scale simulations, there are event streaming techniques which will be essential, as every snapshot can't be saved. Steering is pervasive; in particular sensors sometimes need to be steered

There is a need to respond to variable rates or load changes such as between peak and non-peak hours. This requires an elastic system capable of dynamic scalability based on the changing rates. This creates interesting programming paradigms involving unbounded data with sliding windows with the need to detect changes between windows.

One needs to characterize workflows better; for example where are computations done are they close to the source of the data or does data stream to the cloud? This is exemplified by comparison of Akamai edge vs Google server farms. There is a well known discussion of three processing locations: sensor (source), fog (computing near source) or cloud (back-end). Which cloud one chooses as location might affect turnaround?

Does the sensor's connectivity facilitate certain processing modes or does it constrain processing modes the answer could be time dependent. Does the processing (cloud) need to be self distributed? One needs to address the case with no cloud connection (at a particular time). What aspects of processing are sensitive to hierarchical (sensor-fog-cloud) programming model; does query processing get decomposed at all levels? What parts of system need to talk to all services/nodes? How much can be done at each level of hierarchy?.

Projects such as large telescopes or accelerators with long timelines need to consider timeline of evolving technologies for streaming data as industry is driving rapid change. One can identify needed functions and components and if one programs to this high level model, it should be easier to incorporate underlying technology change.

II. Current State of Technology

Streaming solutions have evolved in both academic research and industry. Not too surprisingly, differences exist between industry and research applications.

The three industry giants – Amazon (Kinesis), Google (Beam) and Microsoft (Trill) have their own streaming technologies. Although operating at large scale their uses like e-commerce and logging transactions have important differences from the research applications discussed. There are many solutions built around open source Apache solutions like Samza, Storm.

Integration of the many technologies to build "end-to-end" streaming infrastructure is a basic challenge, as is the lack of consensus as to appropriate hardware and software infrastructure.

The issues at the intersection of streaming and HPC were a consistent theme across both workshops. One reason to explore this intersection is that commericial solutions are not converging with the needs of science big data. Another is that resources available for large scale research applications are typified as resources configured as HPC clusters and thus with the dual if not conflicting roles of supporting streaming requirements along with static batch-queue requirements.

Industrial solutions are powerful and scalable, but they are designed around a different kind of scalability: the ability to support a large number of applications each of which is not very data intensive. In contrast, scientific applications are characterized by either high data-volumes, rates or latency sensitivity. Further, there is a disconnect between HPC and BigData software stacks in the areas of performance and architectural components.

Other differences pertains to event size, which is often large in research use cases. Nonetheless it is instructive to look at industrial streaming solutions have emerged, including Apache Spark, Flink, Storm, Heron, Samza, Kinesis and MillWheel. We provide a detailed description of the two solutions Heron and Apache Beam that have emerged in as leading industry solutions.

III. Workshop Recommendations

The full list of findings and recommendations can be found at http://streamingsystems.org and workshop report at the ASCR site [1]. Here we provide a selective set of findings and recommendations to motivate a research program in streaming systems for large-scale computational science.

- **Managing the end-to-end workflow is critical to ensure innovations from streaming data.** This requires investigation of new programming models, as the programming and runtime requirements of streaming applications are different from the traditional MPI-based applications that have dominated scientific computing. As such, there is a gap in capabilities of existing programming models. There is a need for a systematic evaluation of existing approaches and research in new programming models. The complexity of the end-to-end orchestration of real-time streaming data and processing imposes performance constraints and new functional requirements. There is a critical need to support real-time steering and human-in-the-loop activities for next- generation streaming workflows.
- **Meeting the needs of the streaming and steering applications will require development of capabilities and support from NSF and DOE high performance computing and networking facilities.** Historically, NSF and DOE HPC facilities have focused on batch queue jobs. Also, the network is a critical part of the streaming and steering dataflows, but is not seamlessly integrated into user workflows. There is a need for better technical support and policies that facilitate streaming data and

steering jobs with batch capabilities. In general, there is a need for community efforts and infrastructure to develop and support capabilities for streaming.

- **Develop and sustain a software ecosystem that supports the needs of the streaming applications on HPC Platforms:** Develop techniques and methodologies that enable streaming solutions to be integrated into existing domain-specific software stacks and ensure long-term software sustainability. Bridge the research versus product tension, and allow for robust prototypes to be integrated into existing application/domain-specific software stacks. In particular, find mechanisms to incorporate advances made by industrial products into scientific computing software ecosystem.

IV. Bringing Streaming to HPC Systems

In response to the recommendations of STREAM workshops, we have an ongoing research agenda. Here we cover three specific aspects – mini-apps, Pilot-Streaming and E-HPC, where the latter two address different limitations arising from traditional static HPC resource management capabilities.

A. Streaming Mini-Applications

We are developing two streaming mini-applications: MASS and MASA.

- MASS (MiniApp for Stream Source): Stream data source, which can be tuned to produce different types of data streams.
- MASA (MiniApp for Streaming Analysis) Stream data analysis/application, which captures most/some of the basic properties of the streaming / online analysis including online reconstruction and other online applications. The purpose of this mini-app is to serve as a benchmark across different runtime systems and infrastructure.

These mini-applications will be used to develop benchmarks that can be used to evaluate existing solutions, identify community best practices and drive generalization across science needs by developing common libraries.

B. Pilot-Streaming: Abstraction based integration resource management for HPC and Stream processing

Stream processing is becoming an increasingly important part of scientific application pipeline. While traditionally streaming applications primarily performance simple analytics (smooth averages, max detection) on the incoming data stream, the computational demands are growing. For example, to run deep learning based computer-vision algorithms, such as convolutional neural networks on the incoming data stream, vasts amounts of scalable compute resources are required.

Thus, the ability to integrate streaming applications with scalable HPC applications becomes increasingly important. Often, streaming, batch and interactive processing utilizing different abstractions and runtime systems need to be combined. Further, batch-based offline algorithms need to be adapted to process windows of data instead of the complete bounded dataset. Often, stream processing frameworks only provide a subset of the functionality of batch processing frameworks and are less mature than these. Building a abstraction that is capable of unifying this disperse landscape of tools is a challenging task.

Having surveyed the current state of streaming frameworks and applications, we propose a novel abstraction for resource management of joint batch and stream processing framework that supports the provisioning of suitable resource configurations and execution strategies for highly dynamic streaming applications. This abstraction – referred to as Pilot-Streaming addresses the challenges identified above in particular with respect to heterogeneity and resource management.

The Pilot-Abstraction offers a unified approach for compute and data management across heterogeneous compute resources (HPC, cloud, Hadoop), storage resources (e. g. local disks, cloud storage, parallel filesystems, SSD) and memory. It provide efficient mechanisms for managing data [2] and compute across different, possibly distributed backends. It allows the efficient management of intermediate and output data taking into account data locality. We have explored the applicability of the Pilot-Abstraction [3] to data-intensive applications on HPC and Hadoop environment [2], [4], [5], [6].

While the original abstraction [**?**] was designed for batch-oriented applications, the addition of streaming capabilities will enhance its applicability and suitability of distributed data-intensive applications. The coupling of data processing with a variety of data sources (often external to the streaming application) using a message broker, such as Kafka. Pilot-Memory allows the coupling of data production and processing within the same resource.

Pilot-Streaming is an extensible framework allowing the simple addition of new streaming data sources and processing frameworks.

- **Streaming Data Processing API and Framework:** For the processing of streaming data applications can utilize the Pilot-API for defining Compute- Units. Compute-Units can subscribe to streaming Data-Units and process these exploiting task-level if necessary.
- **Streaming Data Source Access:** Make streaming source (e. g. Kafka topics) accessible for Compute-Units and enable the micro-batch processing of discretized chunks of incoming data. The Kafka adaptor is implemented using the Kafka Python API [7].
- **Extensibility and Interoperability:** In addition to the Pilot-API, Pilot-Streaming support the interoperable use of other stream processing frameworks (e. g. Spark Streaming and Flink), which enables applications to utilize the different capabilities of these frameworks in a unified way. By generalizing the window function abstraction into a higher- level abstraction, applications can be expressed independently of the underlying execution engine.

Figure 1 illustrates how we utilize the Pilot-Abstraction to provide runtime for streaming applications as well as to support the interoperable use of third-party streaming engines where appropriate.

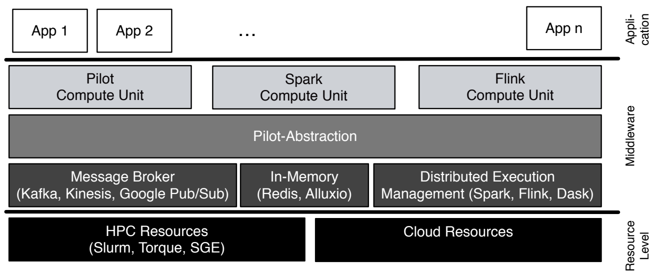
App 1
App 2
...
App n
Appli-cation
Pilot Compute Unit
Spark Compute Unit
Flink Compute Unit
Pilot-Abstraction
Middleware
Message Broker (Kafka, Kinesis, Google Pub/Sub)
In-Memory (Redis, Alluxio)
Distributed Execution Management (Spark, Flink, Dask)
HPC Resources (Slurm, Torque, SGE)
Cloud Resources
Resource Level

Fig. 1. Pilot-Job: Streaming and Memory Capabilities Extension

In this paper, we demonstrate (i) the extension of the Pilot-Data capabilities to the management of unbounded streaming data, (ii) how application-level scheduling mechanism provided by the Pilot-Abstraction can interoperate with distributed execution engines designed for streaming analytics (such as Spark Streaming). We will use MASS and MASA to validate the proposed architecture end-to-end from the streaming data source to the processing layer.

C. E-HPC: Supporting elasticity for Stream processing

Science experiments are increasingly processed on HPC systems as complex scientific workflows with real-time and dynamic resource needs. However, as was highlighted in the workshops findings, today's HPC platforms are still designed to support monolithic static MPI applications and present severe challenges in performance, utilization and reliability for next-generation scientific workflows that need elastic management of HPC resources. There is a need for an elastic framework to support the dynamic and real-time needs of scientific streaming workflows running on HPC resources.

In our current work, we are developing an elastic framework, **Elastic-HPC (E-HPC)** for managing resources of scientific workflows in an HPC environment. It provides a dynamic, adaptable resource management and execution framework that is capable of growing and shrinking the allocated resources for a workflow during execution. Specifically, existing workflows and workflow tools can use E-HPC as a backend execution framework. E-HPC transforms the HPC resource substrate to a malleable platform for stream processing pipelines.

E-HPC manages an elastic dynamic window of resources for stream processing pipelines built over todays static HPC jobs. E-HPC handles the dynamic resource needs of stream processing by auto-scaling. In its current implementation, it uses checkpoint-restart mechanism to save and launch workflow stages on different number of resources and interfaces with the workflow engine to manage the interaction with the user and managing the execution. Users or their scripts scripts or workflow programs can setup appropriate policies, rules on managing the resources for the dynamic streams. Users can either submit a workflow description to E-HPC or instrument their workflow script through the interface that E-HPC provides.

E-HPC has two main components – i) coordinator and ii) tracker. The coordinator interfaces with the users and workflow engines, whereas the tracker keeps track of job execution and resource requirements. A user submits a workflow to E-HPC, and the coordinator generates job submission scripts corresponding to the different stages of the workflow. The jobs are submitted to run on HPC resources through the batch scheduler. E-HPC also creates a tracker for each job that runs on these HPC resources. It uses Distributed MultiThreaded CheckPointing (DMTCP), a checkpoint-restart library, for saving the execution state of workflow stages and restart the workflow on a different set of resources. This auto-scaling capability of E-HPC allows a streaming workflow to dynamically adapt to its changing data and resource requirements.

References

[1] https://science.energy.gov/∼/media/ascr/pdf/programdocuments/docs/2017/STREAM2016.pdf.

[2] A. Luckow, M. Santcroos, A. Zebrowski, and S. Jha, "Pilot-data: An abstraction for distributed data," *Journal of Parallel and Distributed Computing*, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514001725

[3] A. Luckow, L. Lacinski, and S. Jha, "SAGA BigJob: An Extensible and Interoperable Pilot-Job Abstraction for Distributed Applications and Systems," in *The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2010, pp. 135–144.

[4] A. Luckow, P. K. Mantha, and S. Jha, "Pilot-abstraction: A valid abstraction for data-intensive applications on hpc, hadoop and cloud infrastructures?" *CoRR*, vol. abs/1501.05041, 2015. [Online]. Available: http://arxiv.org/abs/1501.05041

[5] A. Luckow, I. Paraskevakos, G. Chantzialexiou, and S. Jha, "Hadoop on HPC: integrating hadoop and pilot-based dynamic resource management," *CoRR*, vol. abs/1602.00345, 2016. [Online]. Available: http://arxiv.org/abs/1602.00345

[6] A. Luckow, I. Paraskevakos, G. Chantzialexiou, and S. Jha, "Hadoop on HPC: Integrating Hadoop and Pilot-based Dynamic Resource Management," *IEEE International Workshop on High-Performance Big Data Computing in conjunction with The 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2016)*, 2016.

[7] A. Montalenti, "Pykafka: Fast, pythonic kafka, at last!" http://blog.parsely.com/post/3886/pykafka-now/, 2016.